1 ***(1,5 puntos)*** *Explique el funcionamiento de la instrucción atómica* `lock.xchg .Ri, /dir` *e implemente mediante la misma un cerrojo con espera activa (spin-lock) para máquinas con memoria compartida.*

SOLUCIÓN

Esta instrucción (*exchange* con el prefijo lock) está presente en algunas arquitecturas para que se puedan construir mecanismos de sincronización de memoria compartida de mayor abstracción, como pueden ser por ejemplo los cerrojos (*locks*) y las barreras (*barriers*).

La instrucción devuelve en Ri el contenido previo de la dirección especificada, `dir`, y escribe el contenido de Ri en esta posición y esto de manera atómica, i.e., con la garantía de que ningún otro procesador pueda acceder a esta misma dirección mientras dura la ejecución de esta instrucción. Este comportamiento lo podemos expresar de la siguiente forma, suponiendo que las acciones comprendidas entre las llaves ocurren atómicamente:

```
lock.xchg  .Ri, /dir
      { tmp  <-- /dir
        /dir <-- Ri
        Ri <-- tmp
      }
```

Una posible implementación de una primitiva para ganar el acceso a un cerrojo con espera activa podría ser, simplemente, tratar de reproducir el comportamiento de una instrucción de `test&set`, tal y como se desarrolló en una de las sesiones de ejercicios de la asignatura. Por simplicidad, podemos suponer que la ejecución de la instrucción afecta a los flags aritméticos según el valor que retorna en el registro Ri y, también, que se trata de una arquitectura genérica que permita los direccionamientos que empleamos en el código.

```
          LD .R1, #D'01
intento:  lock.xchg .R1, /dir_cerrojo
          bnz $intento
          ret
```

donde se debe notar que no haría falta volver a llevar un uno al registro empleado en cada vuelta (o *spin*) (puesto que devuelve ese valor en el registro R1) y que una implementación más eficiente sería la que hiciese primero la comprobación en lectura de que el cerrojo está abierto y luego intentase realizar su cierre en acceso atómico, lo que denominamos *test-and-test-and-set*.

2 ***(1 punto)*** *Indique en qué consiste la variante de la política de coherencia de cachés mediante invalidación conocida como Read_Broadcast y en qué medida puede reducir el número de fallos por invalidación. Explique hasta qué punto sería útil en una situación en que existieran múltiples escritores y lectores accediendo a un mismo bloque.*

SOLUCIÓN

Si se aplica una política de invalidación "pura", el acceso a cada una de las copias de un mismo bloque que ha sido modificado por un procesador producirá un fallo (los llamados fallos por invalidación o *invalidation misses*). Con la variante *Read_Broadcast* se reduce a uno el número de este tipo de fallo por cada invalidación que se realice, puesto que se aprovecha la lectura del bloque actualizado por parte del primer procesador que se lo encuentre en estado no válido para actualizar las demás copias.

Esta variante de la política de invalidación resulta ideal en el caso de bloques a los que se acceda por parte de un solo escritor y varios lectores, y se aleja de ese ideal conforme aumenta el número de procesadores que escriben en él, llegando al extremo, donde todos los procesadores escriben en un patrón temporal de grano fino, a no representar ninguna mejora con respecto a la política de invalidación pura.

3 ***(2,5 puntos)*** *Sea un sistema UMA en que se emplea un mecanismo de coherencia de cachés con implementación snoopy y el protocolo, basado en invalidación, MESI.*

a) *Describa los estados en que puede estar un bloque de memoria caché.*

En este sistema se suceden los tres accesos siguientes y en el mismo orden en que aparecen:

b) *Un procesador* **i** *accede en lectura a su copia local en caché de la variable* `A` *cuyo bloque correspondiente se encuentra en estado no válido (I). Suponiendo que dicho bloque está en la caché del procesador* **j** *en estado compartido (S), indique la secuencia de acciones que se da en el sistema para obtener el valor actualizado de* `A`*. Señale qué modificaciones se producen en el sistema.*

c) *A continuación el procesador* **i** *escribe un nuevo valor en* `A`*. Indique la secuencia de acciones a que da lugar esta escritura y las modificaciones que conlleva en el sistema.*

d) *En un instante posterior, el mismo procesador* **j** *accede en lectura a la variable* `A`*, cuyo bloque correspondiente* no *ha sido reemplazado de su caché. Indique la nueva secuencia de acciones a que da lugar esta lectura y las modificaciones que se producen.*

SOLUCIÓN

a) Podríamos resumir los estados del modo siguiente. Hay dos estados, digamos, "extremos", el estado I (Invalid), en que el bloque no es válido y el estado M (Modified), en que es la única copia válida en toda la máquina (y que en su momento dará pie a que se actualice en memoria principal y/o a servir el fallo por invalidación que se dé en otro procesador).

Los otros dos son el estado E (Exclusive), que representa una especie de paso previo al estado M (el bloque es también la única copia en caché, pero es coherente con la copia de memoria, que por lo tanto no necesitaría ser actualizada en *copy-back*), y el S (Shared), en que el bloque está potencialmente copiado en otras cachés y también es coherente con la copia de memoria. En este estado las escrituras se deberán difundir por el bus para que esas otras copias se invaliden.

b) Se tratará de un **fallo en lectura**, puesto que la copia del bloque aparece en estado I, no válido. Si el bloque está en estado S en j, podría estarlo en otros procesadores también. Supondremos que es j quien suministra la copia válida del bloque. Se darán los siguientes pasos:

1. el procesador i hace una petición a memoria a través del bus
2. la caché j (que ha detectado la petición haciendo *snooping*) pone el valor en el bus (que se arbitra)
3. se abandona el acceso a memoria
4. el procesador i copia en su caché el valor del bus y se pone en estado S
5. todas las otras copias (incluida la de j) permanecen en estado S

c) Se trata de un **acierto en escritura**, ya que el bloque se encuentra en estado S. Se darán los siguientes pasos:

1. el procesador i difunde en el bus una operación de invalidación
2. los procesadores (haciendo *snooping*) con copia en estado S (entre los que se encuentra seguro j, supuesto que el bloque no haya sido reemplazado localmente) transitan: S->I (no válido)
3. se actualiza el valor local en la caché
4. cambio en el estado local en i: S->M (modificado, única copia válida)

4 ***(5 puntos)*** *Suponga el siguiente fragmento de código escrito para un computador secuencial.*

```
(1)          add r1, r0, r0      ; i=0              for (i=0; i<N; i++)
(2)   bi:    add r2, r0, r0      ; j=0                 for (j=0; j<M; j++){
(3)   bj:    st r0, #0(r11)      ; mA(i,j)=0              mA[i][j] = 0;
(4)          ld r3, #0(r10)      ; r3=mB(i,j)             if (mB[i][j] != 0)
(5)          beqz r3, $noalm     ;                           { mA[i][j] = mB[i][j] && masc;
(6)          and r3, r3, r8      ;                             nocero++;
(7)          st r3, 0(r11)       ; mA(i,j)=r3              }
(8)          add r4, r4, #1      ; nocero++             }
(9)  noalm:  add r10, r10, #4
(10)         add r11, r11, #4
(11)         add r2, r2, #1      ; j++
(12)         cmp r15, r2, r20
(13)         blt r15, $bj        ; si j<M ir a bj
(14)         add r1, r1, #1      ; i++
(15)         cmp r15, r1, r21
(16)         blt r15, $bi        ; si i <N ir a bi
(17)         add r2, r0, r0      ;                        for (j=0 .....)
```

Suponga que el procesador en el que se ejecuta este programa está segmentado en 5 etapas de pipeline. Este procesador no dispone de mecanismos de adelantamiento, e inserta ciclos de parada para resolver tanto las dependencias de datos como las de control. Las tareas realizadas en cada etapa de pipeline son:

- *E1: Fetch e incremento del PC*
- *E2: Decodificación, lectura de registros, evaluación de la condición y cálculo de la dirección destino en las instrucciones de salto.*
- *E3: Ejecución y cálculo de la dirección para las instrucciones load y store.*
- *E4: Lectura o escritura de datos en memoria*
- *E5: Escritura en registros*

a) *Indique los distintos tipos de dependencias que producen parones en la ejecución del código del enunciado, así como el número de ciclos de espera que tiene que introducir el procesador para resolverlas. Razone su respuesta.*

b) *Suponga ahora que, manteniendo las mismas etapas, el procesador dispone de salto retardado y de todo tipo de mecanismos de adelantamiento.*

b.1) *Indique en cuánto se reducen los ciclos de parada gracias a la utilización de los mecanismos de adelantamiento. Especifique qué tipos de adelantamiento serían necesarios. Razone su respuesta.*

b.2) *Indique si el programa se ejecutaría correctamente, al disponer el procesador de salto retardado, y de no ser así indique la solución más sencilla.*

b.3) *Reordene el fragmento de código para eliminar* ***únicamente*** *los ciclos de parada debidos a las dependencias de datos y calcule el CPI que se obtendría para N=100 y M=500, suponiendo que el 40 % de los elementos de mB tuviese el valor cero.*

c) *Sabiendo que el tiempo medio de ejecución por instrucción es de 8,5 ns en el computador sin pipeline, calcule la ganancia que se obtendrá con el computador con pipeline y el código reordenado. Calcule además la productividad (throughput), expresada en MIPS, que se obtendría en cada caso. Suponga para ello que el tiempo de ciclo del pipeline es 2 ns.*

SOLUCIÓN

a) Las dependencias de datos que producen parones son las siguientes:

Entre las instrucciones (4)-(5), (6)-(7), (11)-(12), (12)-(13), (14)-(15), y (15)-(16).

En todos los casos, el procesador debe introducir 3 de ciclos de espera para resolverlas puesto que la escritura en el banco de registros se produce en la etapa 5 mientras que la lectura se realiza en la etapa 2, y no se indica en el enunciado que el ciclo de acceso al banco de registros este dividido en subciclos.

Las dependencias de control se dan en las instrucciones (5), (13) y (16). En este caso, el procesador debe introducir 1 ciclo de espera tras cada instrucción de salto ya que hasta la etapa 2 no se conoce la dirección de la instrucción destino del salto ni si se cumple o no la condición en los saltos condicionales. De este modo se evita que se ejecuten las instrucciones secuenciales.

b) La utilización de los mecanismos de adelantamiento produce los efectos siguientes:

Elimina totalmente los ciclos de espera entre las instrucciones (6)-(7) (adelantamiento $E3- > E4$), (11)-(12) y (14)-(15) (adelantamiento $E3- > E3$)

Reduce de 3 a 2 los ciclos de espera entre las instrucciones (4)-(5) (adelantamiento $E4- > E2$), y reduce de 3 a 1 los ciclos de espera entre las instrucciones (12)-(13) y (15)-(16) (adelantamiento $E3- > E2$).

b.1) El programa no se ejecutaría correctamente ya que siempre se ejecutaría la instrucción siguiente a una de salto puesto que tanto la condición como la dirección destino de los saltos se calcula en la etapa E2. La solución más sencilla es introducir una instrucción NOP detrás de las instrucciones 5, 13 y 16.

b.2) Una posible solucion de reordenado del código, con la que se consigue disminuir los ciclos de espera debidos a las dependencias de datos es la siguiente:

```
(1)          add r1, r0, r0      ; i=0
(2)   bi:    add r2, r0, r0      ; j=0
      bj:
(4)          ld r3, #0(r10)      ; r3=mB(i,j)
(3)          st r0, #0(r11)      ;
(9)          add r10, r10, #4
(5)          beqz r3, $noalm     ;
             NOP
(6)          and r3, r3, r8      ;
(7)          st r3, 0(r11)       ; mA(i,j) = r3
(8)          add r4, r4, #1      ; nocero++
   noalm:
(11)         add r2, r2, #1      ; j++
(12)         cmp r15, r2, r20
(10)         add r11, r11, #4
(13)         blt r15, $bj        ; si j<M ir a bj
             NOP
(14)         add r1, r1, #1      ; i++
(15)         cmp r15, r1, r21
(17)         add r2, r0, r0      ; j=0
(16)         blt r15, $bj        ; si i <N ir a bj
             NOP
```

Cálculo del CPI:

- Instrucciones ejecutadas:
  1 + 1 + 100x4 + 100x500x(0,6x11 + 0,4x8) = 490.402
- Ciclos empleados en la ejecución:
  4 + 1 + 1 + 100x(4+1nop) + 100x500x(0,6x11 + 0,4x8 + 2nop) = 590.506

CPI = (590.506-4)/490.402 = 1,2

c) Partiendo del tiempo de ejecución y del número de instrucciones ejecutadas, calculados en el apartado anterior, y dado que el tiempo de ciclo del pipeline es 2 ns, el tiempo medio de ejecucion por instruccion en el computador con pipeline es:

$$\frac{590.506\ ciclos \times 2ns/ciclo}{490.402\ inst} = 2,4ns$$

La ganancia obtenida seria por lo tanto de 8,5/2,4 = 3,54.

La productividad, en el computador sin pipeline sería de $1/8{,}5\text{x}10^{-9}$ instrucciones por segundo, es decir 117,65 MIPS, mientras que en el computador con pipeline sería de $1/2{,}4\text{x}10^{-9}$ instrucciones por segundo, o lo que es lo mismo 416,6 MIPS.

1 [1,5 puntos] Responda breve y *razonadamente* a las siguientes cuestiones relativas a los multiprocesadores (MP) y a la computación de altas prestaciones, HPC:

a) ¿Cuál es el orden de magnitud del rendimiento de los sistemas en los primeros puestos del Top500?

b) ¿Cuál es la ganancia o *speedup* ideal en la versión paralela de un programa ejecutado sobre p procesadores o núcleos?

c) ¿Qué establece la Ley de Amdahl?

d) ¿Qué significa el coste u *overhead* asociado al paralelismo?

SOLUCIÓN

a) Los sistemas más potentes de este ránking, Top10, superan todos ellos el PFLOPS (petaflops, 1.024 TFLOPS), valor que fue batido por primera vez en 2008 por el sistema *Roadrunner* de IBM. Las máquinas situadas en los últimos puestos de esta clasificación tienen capacidad de ejecución del orden de decenas de TFLOPS.

b) En una paralelización ideal con todo el trabajo potencialmente mejorable y sin sobrecoste (*overhead*) el tiempo de ejecución original, t, se repartiría uniformente entre los p procesadores o núcleos, obteniéndose un tiempo de t/p y un speedup de p, speedup(p) = p

c) En esta ley se dice que la ganancia que se obtiene al paralelizar un programa con p procesadores está limitada por la proporción de su tiempo de ejecución (1, normalizado) en la que se puede realizar está mejora, 1-S, siendo S la fracción no mejorable. Se obtiene, por tanto, que el speedup máximo posible sería 1/(S+(1-S)/p). En esta expresión se observa que según aumenta p el tiempo resultante tiende asintóticamente a S.

d) La creación de un proceso o tarea, el coste de comunicación entre procesadores, la actividad que se genera para mantener la coherencia de las cachés de los procesadores, etc., son ejemplos de que la versión paralela de un programa conlleva ineludiblemente ciertos sobrecostes en tiempo o "peajes" (para lo que se suele utilizar el término inglés *overhead*) que son inherentes a la ejecución paralela y que no aparecían en la versión secuencial del programa. Estos sobrecostes pueden llegar a hacer que la versión paralela tenga un tiempo de ejecución peor que la versión secuencial y, por tanto, deben tenerse siempre en cuenta a la hora de la paralelización.

2 [1,5 puntos] Para una arquitectura multiprocesador de memoria compartida, indique de manera razonada si funciona correctamente la siguiente implementación del acceso a un cerrojo (que se encuentra en la dirección común a todos dir_cerrojo) mediante espera activa o *spin_lock*.

```
lock: ld .R1, /dir_cerrojo
      cmp .R1, #0
      bnz $lock
      st #1, /dir_cerrojo
      ret
```

SOLUCIÓN

Si se analiza el código, se observa que tiene una estructura de bucle sobre el valor que se lee de la dirección del cerrojo hasta que éste devuelve el valor 0. Esta lectura desde memoria se realiza con una instrucción `load`, y luego se ejecutan dos instrucciones más, `cmp` y `bnz`, para comprobar el valor leído y en su caso realizar el salto hacia atrás, de nuevo al load.

En principio, este código parece comportarse de acuerdo a lo esperado: espera activa hasta comprobar que el cerrojo está liberado o abierto. Sin embargo, hay que tener en cuenta que en su ejecución paralela no es *atómica* la secuencia que resulta de la lectura del valor de la dirección, su posterior comprobación y, en su caso, almacenamiento de un uno para su cerrado. Por lo tanto, diferentes procesadores podrían ganar el acceso si no se garantiza que desde el instante que leen hasta que escriben ningún otro procesador puede hacerlo, lo que constituye un clásico ciclo de RMW, lectura-modificación-escritura, y que se suministra por las arquitecturas de los procesadores con diferentes instrucciones (test&set, load-locked/store-conditional, lock.exchange, etc.)

3 [1,5 puntos] Indique las ventajas y defectos de las políticas de invalidación y actualización para mantener la coherencia de cachés. ¿Qué política cree que se comportaría mejor en el caso del acceso a un bloque que contuviese un cerrojo al que se accede en espera activa o *spin_lock*?

SOLUCIÓN

En el caso de invalidación, en general, se disminuye el tráfico extra necesario para mantener la coherencia, y resulta ideal en una compartición secuencial o de grano gordo en la que durante un intervalo grande de tiempo sólo accede un procesador a un determinado bloque. Sin embargo, el número de fallos de cache se acumulará al existente en un comportamiento monoprocesador y que corresponde a la no presencia de un determinado bloque. Estos fallos acumulados darán lugar a que se demande una copia válida del bloque que ha producido el fallo y, por tanto, también añadirán tráfico en pro de la coherencia en el sistema. Dependiendo del patrón de acceso al bloque, este tráfico extra puede llegar a superar al que se genera en la política de actualización, en la que siempre que un bloque esté compartido se difunde su modificación para que el resto de copias se mantengan coherentes.

La ventaja fundamental de esta segudna política es que no se producen fallos añadidos debidos a la política de coherencia y que, en principio, todas las copias son siempre válidas, por lo que una secuencia de accesos en que diferentes procesadores accediesen en instantes próximos en el tiempo al mismo bloque (compartición que hemos llamado de grano fino) se comportaría de manera más adecuada que la invalidación, donde cada acceso produciría un fallo y la correspondiente demanda del bloque (este efecto se aliviaría con la variante que aparece en la pregunta anterior).

El caso de acceso a un cerrojo en espera activa, si ésta se realiza en su versión más rudimentaria con un test&set que constantemente consulta y modifica el bloque correspondiente (tal y como aparece en el enunciado de la pregunta siguiente) y con un patrón de acceso en que varios procesadores compitan pro ganar el acceso, parece más adecuado el uso de la actualización, puesto que de lo contrario cada procesador, en cada "vuelta" al spin del cerrojo, produciría fallo de cache y la consiguiente demanda del bloque modificado, que ya se hubiera obtenido en el caso de emplear actualización.

4 (5,5 puntos) Suponga el siguiente fragmento de código escrito para un computador secuencial.

```
(1)buc:    ld   r1, #0(r11)        ; for (i=0; i<N; i++) {
(2)        ld   r2, #0(r12)        ;    c[i] = a[i] + b[i];
(3)        add  r3,  r1, r2        ;    d[i] = 2*a[i];
(4)        st   r3, #0(r13)        ; }
(5)        add  r1, r1, r1         ; j = 256;
(6)        st   r1, #0(r14)        ; ........
(7)        sub  r4, r4, #1
(8)        bz   r4 $salir
(9)        add  r11, r11, #4
(10)       add  r12, r12, #4
(11)       add  r13, r13, #4
(12)       add  r14, r14, #4
(13)       br   $buc
(14)salir: add  r4, r0, #256
           ...........
```

Suponga que el procesador en el que se ejecuta este programa está segmentado en 5 etapas de pipeline. Este procesador no dispone de mecanismos de adelantamiento, e inserta ciclos de parada para resolver los distintos tipos de dependencias. Las tareas realizadas en cada etapa de pipeline son:

- E1: Fetch
- E2: Decodificación, lectura de registros y cálculo de la dirección de salto
- E3: Ejecución, evaluación de la condición de salto y cálculo de dirección en las instrucciones load y store.
- E4: Acceso a memoria
- E5: Escritura en registros

a) Determine los distintos tipos de dependencias que se dan en la ejecución del programa anterior, así como los ciclos de parada que introducirá el procesador para resolverlos, y calcule el CPI que se obtendrá para N=1000.

b) Suponga ahora que, manteniendo las mismas etapas, el procesador dispone de todo tipo de mecanismos de adelantamiento, y que para los saltos condicionales utiliza predicción dinámica de salto con 1 bit de historia. La predicción se realiza en la etapa E1 y la comprobación de si se acierta o se falla en la predicción se hace en la etapa E3.

b.1) Determine de forma razonada en cuánto se reducen los ciclos de parada gracias a la utilización de los mecanismos de adelantamiento.

b.2) Determine de forma razonada en cuánto se reducen los ciclos de parada gracias a la utilización de la predicción de salto. Suponga para ello que la primera vez que se ejecuta la instrucción de salto condicional se falla en la predicción.

b.3) Reordene el fragmento de código para conseguir el mejor CPI y calcule su nuevo valor, así como la ganancia que se obtendría con respecto a los resultados obtenidos para el computador del apartado a).

SOLUCIÓN

a) Teniendo en cuenta que únicamente se pueden producir dependencias de datos y de control, las dependencias y los parones correspondientes a dichas dependencias son los siguientes:

- Dependencias de datos. Hay cuatro dependencias de datos RAW que dan lugar a ciclos de parada: entre las instrucciones 2-3, 3-4, 5-6 y 7-8. Para resolver estas dependencias, el procesador introducirá 3 ciclos de parada, ya que la escritura en el banco de registros se produce en la etapa E5 y la lectura en la etapa E2.
- Dependencias de control. Hay dos dependencias debidas a las instrucciones 8 y 13. Para resolver la dependencia producida por la instrucción 8 el procesador introducirá 2 ciclos de parada puesto que, aunque la dirección de salto se calcula en la etapa E2, la condición se comprueba en la etapa E3. En cambio, para resolver la dependencia producida por la instrucción 13, puesto que es un salto incondicional, el procesador sólo introducirá 1 ciclo de espera ya que la dirección de salto se calcula en la etapa E2.

Teniendo en cuenta lo anterior, y para N=1000, el CPI que se obtendrá es:

$$CPI = (999 \times (13 + 12\ pdd + 3\ pdc) + (8 + 12\ pdd + 2\ pdc) + 1)\ ciclos/(999 \times 13 + 8 + 1)\ inst. =$$

$$= 27.995\ ciclos/12.996\ inst. = 2,15\ ciclos/inst.$$

donde pdd indica los parones debidos a dependencias de datos y pdc los debidos a dependencias de control.

b) Teniendo en cuenta que el procesador tiene todo tipo de mecanismos de adelantamiento y predicción dinámica para los saltos condicionales.

b.1) Los mecanismos de adelantamiento hacen que se eliminen los ciclos de espera debidos a las dependencias de datos entre las instrucciones 3-4, 5-6 (Alu-Memoria), 7-8 (Alu-Evaluación Condición). Con respecto a la dependencia entre las instrucciones 2-3, se reducen de 3 a 1 los ciclos de espera, gracias al adelantamiento Memoria-Alu.

Dado que el número de ciclos de espera sin adelantamientos era 1.000x12 = 12.000 y con adelantamientos es 1.000x1 = 1.000, se produce una reducción de 11.000 ciclos en la ejecución del código.

b.2) La utilización de predicción dinámica hace que únicamente se produzcan ciclos de espera cuando se falla en la predicción. Dado que esta comprobación se realiza en la etapa E3, se introducirán en este caso 2 ciclos de espera. Este mecanismo se utiliza en la ejecución de la instrucción 8, produciéndose 2 fallos de predicción; un fallo la primera vez que se ejecuta (según se indica en el enunciado), y otro la última (el bit de historia tiene el valor 0, se predice no saltar, pero se salta a la instrucción 14).

Dado que el número de ciclos de espera debidos a la ejecución de la instrucción 8 sin predicción era 1.000x2 = 2.000 y con predicción es 2x2 = 4, se produce una reducción de 1.996 ciclos en la ejecución del código.

b.3) Para conseguir el mejor CPI hay que reordenar el código de modo que se elimine el ciclo de espera debido a la dependencia de datos entre las instrucciones 2-3. Una posible solución sería colocar la instrucción 7 entre ambas instrucciones. Con ello se eliminarían 1.000 ciclos de espera.

Teniendo en cuenta todo lo anterior el CPI que se obtendrá es:

$$CPI = (27.995 - 11.000 - 1.996 - 1.000)\ ciclos/12.996\ inst. = 13.999/12.996 = 1,078\ ciclos/inst.$$

por lo que la ganancia final obtenida sería:

$$G = 2,15/1,078 = 1,99$$

1 (2 puntos) Durante una sesión de medida de media hora, un monitor software ha extraído las siguientes variables operacionales básicas de un servidor web:

Variable	Valor
A	360 peticiones
C	351 peticiones
B	23 minutos

a) ¿A qué parámetros corresponde cada una de ellas?

b) Razone si existe equilibrio de flujo de trabajos durante el intervalo de observación.

c) A partir de la información anterior calcule las siguientes variables operacionales deducidas del servidor web: tasa de llegada, productividad, utilización y tiempo medio de servicio. No olvide indicar las unidades de cada variable.

SOLUCIÓN

a) Siguiendo la notación utilizada en análisis operacional, las variables obtenidas durante el intervalo de observación corresponden a los parámetros:

A: número de llegadas

C: número de trabajos completados

B: tiempo de ocupación

b) Puesto que, durante el intervalo de observación, el sistema no es capaz de completar todos los trabajos que llegan, no existe equilibrio de flujo de trabajos.

c)

Tasa de llegada: $\lambda = \frac{A}{T} = \frac{360\ peticiones}{1800\ s} = 0,2\ peticiones/s$

Productividad: $X = \frac{C}{T} = \frac{351\ peticiones}{1800\ s} = 0,195\ peticiones/s$

Utilización: $U = \frac{B}{T} = \frac{1380\ s}{1800\ s} = 0,77\ (77\ \%)$

Tiempo medio de servicio: $S = \frac{B}{C} = \frac{1380\ s}{1351\ peticiones} = 3,93\ s/peticion$

2 (1 punto) En un sistema informático se sustituye su unidad de disco por una nueva 5 veces más rápida que la original.

a) ¿Qué fracción de mejora tiene esta unidad de disco si al hacer la sustitución se ha observado una ganancia global del rendimiento de 2?

b) ¿Sería posible que la ganancia observada hubiese sido 6?

c) Razone cuál es el valor máximo de la ganancia y con qué utilización del disco se obtendría.

SOLUCIÓN

a) Aplicando la ley de Amdahl, la fracción de tiempo en que se emplea la mejora (fracción de mejora), es:

$$f = \frac{ganancia\ mejora \times (ganancia - 1)}{ganancia \times (ganancia\ mejora - 1)} = \frac{5 \times (2-1)}{2 \times (5-1)} = 0,625\ (62,5\ \%)$$

b) Es imposible obtener una ganancia global de 6, ya que el componente mejorado lo hace en un valor menor, en concreto de 5.

c) El valor máximo de la ganancia será por tanto de 5, siempre que la unidad de disco se utilizase durante el 100 % del tiempo.

3 (3 puntos) Dada la secuencia de instrucciones que se muestra a continuación:

```
(1)   divf  f10, f0,  f2
(2)   multf f4,  f10, f8
(3)   addf  f8,  f2,  f0
(4)   subf  f10, f6,  f8
```

a) Indique, a la derecha del código, los distintos tipos de dependencias de datos existentes.

b) Indique la evolución de la ejecución de las **tres primeras** instrucciones, utilizando el algoritmo de Tomasulo, en la hoja adjunta.

c) Explique brevemente cuál es la diferencia más importante entre la ejecución de esta secuencia de instrucciones utilizando el algoritmo de Tomasulo y utilizando un pipeline con múltiples unidades funcionales que no lo utilice. ¿Se tardaría en ambos casos el mismo tiempo en ejecutar dicha secuencia de instrucciones? ¿El orden de finalización de las instrucciones sería el mismo?

SOLUCIÓN

a) Los tipos de dependencias de datos existentes son los siguientes:

RAW entre (1) y (2) y entre (3) y (4)

WAR entre (2) y (3)

WAW entre (1) y (4)

b) Véase la hoja adjunta

c) Utilizando el algoritmo de Tomasulo, la instrucción (3) pasa a ejecutarse sin tener que esperar a que se resuelva la dependencia de datos RAW entre (1) y (2). Por el contrario, si no se utilizase dicho mecanismo, la instrucción (3) no podría pasar a la etapa de decodificación, y por lo tanto no podría ejecutarse hasta que se resolviera dicha dependencia. Por todo ello, el tiempo de ejecución sería menor utilizando el algoritmo de Tomasulo, aunque el orden de finalización sería distinto, finalizando primero la instrucción (3), a continuación la (1) y finalmente la (2), mientras que sin utilizar dicho mecanismo las instrucciones finalizarían en el mismo orden en que aparecen en el código.

4 (4 puntos) Suponga el siguiente fragmento de código escrito para un computador sin pipeline.

```
(1)        add r1, r0, r0
(2)        add r4, r0, #100   ;  for (i=0; i<100; i++)
(3) for:   ld r5, #0(r10)     ;       a[i] = 2*a[i]
(4)        sll r5, r5, #1
(5)        st r5, #0(r10)
(6)        add r10, r10, #4
(7)        add r1, r1, #1
(8)        sub r3, r4, r1
(9)        bnz r3, $for       ; $ end for
(10)       add r1, r0, r0     ; for (i=0 .......)
             . . . . . .
```

Se quiere ejecutar este mismo programa en un computador con un pipeline de instrucciones de 5 etapas y tiempo de ciclo de 2 ns, que además dispone de todo tipo de mecanismos de adelantamiento (*forwarding*) y utiliza bifurcación retardada. Las tareas realizadas en cada etapa son las que se muestran a continuación:

- E1: Fetch
- E2: Decodificación, lectura de registros, evaluación de la condición y cálculo de la dirección de salto
- E3: Ejecución y cálculo de la dirección en los accesos a memoria para datos.
- E4: Lectura o escritura de datos en memoria.
- E5: Escritura en registros

a) Indique, de forma razonada, las dependencias de datos que producen parones, así como el número de ciclos de parada que debe introducir el procesador para resolverlas en cada caso.

b) Indique de forma razonada si el código se ejecutaría correctamente, y en caso contrario proponga una solución.

c) Reordene el fragmento de código de modo que se produzca el mínimo número de parones posible. Escriba el código reordenado a la derecha del original. Calcule el tiempo de ejecución del código reordenado así como el tiempo medio de ejecución por instrucción.

d) Sabiendo que el tiempo medio de ejecución por instrucción en el computador sin pipeline es de 8,5 ns, calcule la ganancia que se obtendría con el computador con pipeline, así como la productividad (*throughput*), expresada en MIPS, que se obtendría en cada caso.

SOLUCIÓN

a) Las dependencias de datos que producen parones son las siguientes:

Entre las instrucciones (3) y (4), ya que la instrucción (3) tiene disponible su resultado en la etapa 4 y la instrucción (4) lo necesita en la etapa 3.

Entre las instrucciones (8) y (9). En este caso la instrucción (8) tiene disponible su resultado en la etapa 3 y la instrucción (9) lo necesita en la etapa 2.

En ambos casos habría que introducir 1 ciclo de parada.

b) Al utilizarse bifurcación retardada y calcularse tanto la condición como la dirección de salto en la etapa 2, entraría en el pipeline, y se ejecutaría completamente la instrucción (10), que modificaría el valor de `r1`. Esto haría que el programa no se ejecutase correctamente. Una posible solución sería introducir una instrucción `NOP` detrás de la instrucción de salto.

c) Una posible solución de reordenado del código, con la que se consigue eliminar todos los ciclos de espera, además de una ejecución correcta, es la siguiente:

```
(1)             add r1, r0, r0
(2)             add r4, r0, #100
(3)    for:     ld r5, #0(r10)
(7)             add r1, r1, #1      ; Reordenada
(4)             sll r5, r5, #2
(8)             sub r3, r4, r1
(5)             st r5, #0(r10)      ; Reordenada
(9)             bnz r3, $for        ; $ end for
(6)             add r10, r10, #4    ; Reordenada
(10)            add r1, r0, r0
                . . . . . .
```

El tiempo de ejecución de este código sería:

$4 \times 2\ ns + 2 \times 2\ ns + 100 \times 7 \times 2\ ns = 1.412\ ns$

d) Partiendo del tiempo de ejecución calculado en el apartado anterior, y dado que el numero de instrucciones ejecutadas es 2+7x100 = 702 instrucciones, el tiempo medio de ejecución por instrucción en el computador con pipeline es:

$1.412\ ns/702\ instrucciones = 2\ ns/$instrucción (una instrucción por ciclo)

La ganancia obtenida sería por lo tanto de $8,5/2 = 4,25$.

La productividad, en el computador sin pipeline sería de 117,6 MIPS ($1/(8,5 \times 10^{-9})$) mientras que en el computador con pipeline sería de 500 MIPS ($1/(2 \times 10^{-9})$).

ATENTA

ARQUITECTURA DE COMPUTADORES (Plan 09)

Examen Extraordinario (13 de julio de 2012)

Teoría/Problemas

A,B) OPERADORES EN PARALELO (CONTROLADORES) D) INHIBICIÓN Y ANIDAMIENTO
C) NUM MÁX OPS. E) VARIOS DISCOS DMA CON VARIOS CONTROL. ETH.

1 [3 puntos] Sea un computador de 32 bits con un procesador capaz de ejecutar 1.000 MIPS y cuya Secuencia de Reconocimiento de Interrupción (SRI) dura 10 ns. Este computador se quiere utilizar como *router* de Ethernet y se le quieren conectar varios controladores de Ethernet con las siguientes características:

- Bloques de tamaño variable entre 64 bytes y 1.536 bytes.
- Velocidad de transferencia: 10^8 bits/s.
- Los registros de datos de los módulos de entrada/salida tienen un tamaño de 32 bits.

a) Calcule cuántos controladores de Ethernet pueden operar en paralelo si operan mediante interrupciones sabiendo que sus Rutinas de Tratamiento de Interrupciones (RTI) ejecutan 50 instrucciones.

b) Justifique qué tamaño deberían tener los registros de datos de los módulos de entrada/salida para que pudieran operar 16 unidades en paralelo. Suponga que las RTIs ejecutan 60 instrucciones.

c) Calcule el número máximo de operaciones de entrada/salida por segundo que pueden realizar los 16 controladores Ethernet con el nuevo tamaño de los registros de datos. Para ello suponga:

- Tamaño de bloque 64 bytes.
- La rutina de programación de una operación de entrada/salida ejecuta 100 instrucciones.
- La rutina de fin de operación ejecuta 50 instrucciones.
- La RTI ejecuta 60 instrucciones.

d) Con el fin de registrar las operaciones de Ethernet, se conecta una unidad de disco duro con las siguientes características:

- Velocidad de transferencia: $20 \cdot 10^6$ bytes/s.
- Registro de datos de 128 bits.

Este computador dispone de un sistema de interrupciones de 3 niveles de prioridad (Alto, Medio y Bajo) que permite anidamiento e inhibición selectiva y realiza la identificación mediante vectorización. Justifique a qué líneas de petición de interrupción conectaría los periféricos.

e) Como un único disco duro no es suficiente para registrar las operaciones de los controladores de Ethernet y no se pueden conectar más mediante interrupciones, se decide operar los discos duros mediante acceso directo a memoria (DMA) con robo de ciclo en ráfaga. Calcule el número máximo de discos que se podrían operar mediante (DMA) simultáneamente con los 16 controladores Ethernet suponiendo que:

- El protocolo de concesión y liberación de los buses dura 4 ns.
- Cada ciclo de acceso a memoria tiene una duración de 2 ns.

SOLUCIÓN

a) Se calcula la capacidad de proceso que requiere el tratamiento de las interrupciones de un controlador Ethernet a partir de su frecuencia de petición:

$$Frecuencia\ de\ petición = \frac{v_{transf}}{Bits\ por\ petición} = \frac{10^8\ bits/s}{32\ bits/petición} = 3.125.000\ peticiones/s$$

Tratar cada interrupción supone realizar la SRI y ejecutar la RTI. Es decir, 10 ns y 50 instrucciones lo que equivale a 60 instrucciones en un procesador de 1.000 MIPS.

$$Instrucciones\ por\ segundo = 3.125.000\ peticiones/s \times 60\ instrucciones/petición = 187'5 \cdot 10^6\ instrucciones/s$$

Se necesita una capacidad de cómputo de 187'5 MIPS para atender las interrupciones de un controlador Ethernet. Por lo tanto, puesto que 1.000 MIPS/187'5 MIPS = 5'33, podrán operar en paralelo un máximo de 5 unidades con este procesador.

b) Para que puedan operar 16 controladores cada uno debería consumir como mucho 62'5 MIPS, es decir, exactamente la tercera parte de lo actual. Para reducir la frecuencia de interrupciones a la tercera parte basta con triplicar el tamaño de los registros de datos. Sin embargo, como las RTIs ejecutan en este caso 10 instrucciones más, hay que reducir la frecuencia de interrupciones aún más.

Por lo tanto en lugar de triplicar, que no sería suficiente, se cuadriplica el tamaño del registro de datos y se calcula la capacidad de proceso necesaria:

$$Frecuencia\ de\ petición = \frac{v_{transf}}{Bits\ por\ petición} = \frac{10^8\ bits/s}{128\ bits/petición} = 781.250\ peticiones/s$$

$$Instrucciones\ por\ segundo = 781.250\ peticiones/s \times (60+10)\ instrucciones/petición = 54'6875 \cdot 10^6\ instrucciones/s$$

Con unos registros de datos de 128 bits, podrían operar 16 controladores Ethernet puesto que su consumo total sería 54'6875 MIPS×16 = 875 MIPS, menor de los 1.000 MIPS disponibles.

c) Se calcula el tiempo de una operación de Ethernet para el mínimo tamaño de bloque:

$$T_{Eth} = T_{prog} + T_{tran} + T_{última-RTI} = 100\ ns + \frac{64\ B \cdot 8\ b/B}{10^8\ b/s} + (10 + 60 + 50)\ ns = 5.340\ ns$$

De modo que un controlador Ethernet podrá realizar:

$$Operaciones\ por\ segundo = \frac{10^9\ ns/s}{5.340\ ns} = 187.265\ operaciones/s$$

Y los 16 simultáneamente:

$$Operaciones\ por\ segundo = 187.265\ operaciones/s \times 16 = 2.996.240\ operaciones/s$$

d) Se calcula la frecuencia de petición de interrupciones del disco duro.

$$Frecuencia\ de\ petición = \frac{v_{transf}}{Bytes\ por\ petición} = \frac{20 \cdot 10^6\ bytes/s \cdot 8\ b/B}{128\ bits/petición} = 1'25 \cdot 10^6\ peticiones/s$$

El disco duro deberá tener mayor prioridad que los controladores de Ethernet puesto que solicita interrupciones con mayor frecuencia. Se podría conectar a la línea de prioridad Media y los 16 controladores de Ethernet a la de prioridad Baja.

e) La capacidad de proceso que requieren los 16 controladores de Ethernet es de 875 MIPS, según se calculó en el apartado b)

Por lo tanto, quedan 125 MIPS para operar los discos duros mediante DMA. La atención de las peticiones de interrupción que, en este caso, notifican el fin de las operaciones pueden ser postergadas pero no los robos de ciclos. Se calcula, cuánto tiempo se detiene al procesador por cada robo de ciclo:

$$Tiempo_{DMA} = 4\ ns + \frac{128\ bits}{32\ bits/ciclo} \times 2\ ns = 12\ ns$$

Es decir cada ráfaga detiene al procesador durante 12 instrucciones. Como la frecuencia de petición de robos de ciclo es igual a la frecuencia de petición de interrupciones del apartado d), cada disco duro resta al procesador $1'25 \cdot 10^6 \times 12 = 15$ MIPS y por lo tanto se podrían operar 125 MIPS/15 MIPS = 8 discos duros.

2 [4 puntos] Se dispone de un procesador con modelo de ejecución registro-registro, direccionamiento a nivel de byte y palabra de 8 bytes, con las siguientes características:

- Memoria virtual de 2^{47} bytes. Para la traducción de direcciones se dispone de 2 TLBs; una para instrucciones y otra para datos, con un tiempo de acceso de 1 ns. En caso de producirse fallo en alguna de ellas se realiza la traducción mediante un mecanismo de tablas de páginas multinivel.
- Memorias caché. Dispone de cachés separadas para instrucciones y datos, con tiempo de acceso de 2 ns, organización asociativa por conjuntos, con 2 bloques por conjunto, y bloques de 4 palabras. El acceso a las TLBs y a las cachés no se solapa.
- Pipeline de instrucciones con todo tipo de mecanismos de adelantamiento, y con las 7 etapas que se indican a continuación (entre paréntesis se muestra el tiempo real que tardan en realizarse las operaciones de cada una de ellas). Todas las instrucciones pasan por todas las etapas, pudiendo no realizarse en alguna de ellas trabajo útil. El procesador resuelve las dependencias de control introduciendo ciclos de espera

  E1: Traducción mediante la TLB de instrucciones e incremento del PC (1ns)
  E2: Acceso a la caché de instrucciones (2ns)
  E3: Decodificación y lectura de registros (1ns)
  E4: Ejecución, comprobación de condición y cálculo de direcciones (2ns)
  E5: Traducción mediante la TLB de datos (1ns)
  E6: Acceso a la caché de datos (2ns)
  E7: Escritura en el banco de registros (1ns)

a) Indique breve y razonadamente cuál es el tiempo de ciclo del pipeline y cuál es la justificación de organizar las operaciones relacionadas con el acceso a memoria en dos etapas separadas: E1 y E2 para el acceso a instrucciones, y E5 y E6 para el acceso a datos.

b) Determine razonadamente el número de ciclos de espera que debe introducir el procesador para resolver las dependencias de control, y las dependencias de datos existentes entre dos instrucciones consecutivas en los siguientes casos (considere únicamente los casos en los que no se produce fallo de TLB ni de caché):

b.1) Las dos instrucciones son aritméticas.

b.2) La primera instrucción es `ld` y la segunda es aritmética.

c) Se quiere diseñar un nuevo procesador similar al anterior y con el mismo sistema de memoria, pero con 5 etapas de pipeline. Para ello se está estudiando la posibilidad de solapar el acceso a las TLBs y a las cachés, por lo que las 5 etapas serían:

E1: Traducción mediante la TLB de instrucciones, acceso a la caché de instrucciones e incremento del PC.

E2: Decodificación y lectura de registros.

E3: Ejecución, comprobación de condición y cálculo de direcciones.

E4: Traducción mediante la TLB de datos y acceso a la caché de datos.

E5: Escritura en el banco de registros.

c.1) Sabiendo que las cachés de que se dispone son de 32 KB, calcule cuál sería el tamaño de página mínimo necesario para que se pudiera realizar el solapamiento.

c.2) Sabiendo que las entradas de las tablas de páginas ocupan 1 palabra y que la tabla de páginas de primer nivel debería ocupar exactamente 1 página, determine el número de niveles de tablas de páginas más adecuado desde el punto de vista de un mayor aprovechamiento de la memoria principal. Indique asímismo los campos en que se descomponen las direcciones virtuales y su longitud.

c.3) Determine el tiempo de ciclo de este nuevo procesador. Calcule el número de ciclos de espera que debería introducir para resolver las dependencias de control, y las de datos en los casos mencionados en el apartado b).

d) A continuación se muestra un fragmento de código, escrito para un computador secuencial, que realiza la búsqueda de un dato numérico, almacenado en r10, en una lista encadenada cuya dirección de comienzo está almacenada en r1.

```
(1)    buc:  cmp r15, r1, #0
(2)          bz  r15, $no_encontrado
(3)          ld  r4, #0(r1)
(4)          cmp r15, r4, r10
(5)          bz  r15, $encontrado
(6)          ld  r1, #8(r1)
(7)          br  $buc
```

d.1) Determine cuál de los dos procesadores proporcionará un mejor CPI en la ejecución de este código en caso de no producirse ningún tipo de fallo. Calcule ambos CPIs si la lista tiene 1.000 elementos y el dato buscado no se encuentra en la lista.

e) Calcule el tiempo máximo que se emplearía en un acceso de lectura a memoria en el procesador de 5 etapas en caso de no producirse fallo de página. Para ello tenga en cuenta que el tiempo de acceso a la memoria principal es de 60ns y que las cachés tienen política de lectura *out of order fetch*. Calcule además el número de ciclos de espera que se introduciría en el pipeline en este caso.

SOLUCIÓN

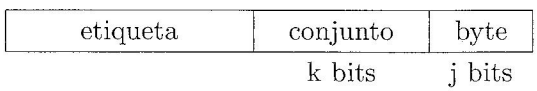
a) El tiempo de ciclo del pipeline es 2ns puesto que viene determinado por la duración de la etapa más lenta. La implementación de las fases de acceso a memoria en 2 etapas separadas está justificada para reducir el tiempo de ciclo ya que si se realizase la traducción en la TLB y el acceso a la caché en la misma etapa éste sería de 3 ns (la suma del tiempo de traducción y el tiempo de acceso).

b) Para resolver las dependencias de control el procesador debe introducir 3 ciclos de espera detrás de todas las instrucciones de salto, ya que el cálculo de la dirección de salto y la comprobación de la condición se realizan en la etapa de ejecución E4. Con respecto a las dependencias de datos entre dos instrucciones consecutivas, y teniendo en cuenta que se dispone de todo tipo de mecanismos de adelantamiento, los ciclos de espera que debe introducir son:

b.1) Dependencia entre dos instrucciones aritméticas: No es necesario introducir ningún ciclo entre la instrucción que genera el resultado y la que lo utiliza como operando fuente debido a la existencia de adelantamiento ALU-ALU.

b.2) Dependencia generada por una instrucción `ld`: El procesador debe introducir 2 ciclos entre dicha instrucción y la siguiente, que lo utiliza como operando, debido a la existencia de adelantamiento MEM-ALU.

c) Las direcciones físicas con las que se accederá a las caches, al ser asociativas por conjuntos, tendrán el siguiente formato:



c.1) Para que sea posible el solapamiento de la traducción en la TLB con el acceso a la Mca, el número de bits utilizados para seleccionar el byte dentro de la página, (información que no se traduce) debe ser mayor o igual que k+j.

Puesto que los bloques son de 32 bytes (4 palabras de 8 bytes), j = 5 bits

Las caches tienen 32 KB/32 bytes/bloque = 1.024 bloques, es decir 512 conjuntos de 2 bloques, por lo que k = 9 bits.

Por lo tanto, el tamaño mínimo de las páginas debe 16 KB (2^{14} bytes).

c.2) Ya que la tabla de primer nivel debe ocupar 1 página, dicha tabla tendrá:

$$2^{14} \text{ bytes/página} \;/\; 2^{3} \text{ bytes/entrada} = 2^{11} \text{ entradas}$$

por lo que serán necesarios 11 bits para direccionar dicha tabla.

Para completar la dirección virtual quedan 47 - (11+14) = 22 bits, a utilizar para el resto de los niveles, por lo que para aprovechar mejor la memoria principal conviene utilizar 11 bits para las tablas de segundo nivel y los otros 11 para tablas de tercer nivel. De este modo todas las tablas ocupan una pagina completa.

Las direcciones virtuales tendrán por lo tanto el siguiente formato:

zona	región	página	byte
11 bits	11 bits	11 bits	14 bits

c.3) AL solaparse el acceso a la TLB y a la caché, el tiempo de las etapas E1 y E4 es el máximo de los tiempos empleados por cada dispositivo, es decir 2ns. Dado que las operaciones de las etapas E2, E3 y E5 tardan 1ns, 2ns y 1ns respectivamente, el tiempo de ciclo del *pipeline* en el nuevo procesador será el tiempo empleado en la etapa más lenta, es decir 2 ns.

Con respecto a los ciclos de espera que debe introducir en este caso el procesador, y siguiendo el mismo razonamiento del apartado b), aplicado ahora al nuevo *pipeline* de 5 etapas:

- Dependencias de control. Ya que tanto el cálculo de la dirección de salto como la comprobación de la condición se realizan en la etapa de E3, el procesador debe introducir 2 ciclos detrás de todas las instrucciones de salto.
- Dependencias de datos entre dos instrucciones aritméticas consecutivas. No es necesario introducir ningún ciclo de espera.
- Dependencias generadas por instrucciones `ld`. Hay que introducir 1 ciclo entre la instrucción que genera el resultado y la que lo utiliza como operando fuente.

d) Para realizar el cálculo del CPI, se indican a continuación los ciclos de espera que se introducen para resolver las dependencias para ambas CPUs, diferenciándose entre los debidos a dependencias de control (dc) y los debidos a dependencias de datos (dd).

```
                                        CPU1     CPU2
(1)  buc:  cmp r15, r1, #0
(2)        bz  r15, $no_encontrado      3dc      2dc
(3)        ld  r4, #0(r1)               2dd      1dd
(4)        cmp r15, r4, r10
(5)        bz  r15, $encontrado         3dc      2dc
(6)        ld  r1, #8(r1)
(7)        br  $buc                     3dc      2dc
```

Los CPIs obtenido en la ejecución del programa, para cada una de las CPUs, si no se produjera ningún tipo de fallo serían los siguientes:

$$CPI_{CPU1} = \frac{1.000 \times (7 + 9dc + 2dd) + (2 + 3dc)}{1.000 \times 7 + 2} = 2,57$$

$$CPI_{CPU2} = \frac{1.000 \times (7 + 6dc + 1dd) + (2 + 2dc)}{1.000 \times 7 + 2} = 2$$

Los $(2 + 3dc)$ y los $(2 + 2dc)$ ciclos adicionales que aparecen en el numerador de ambas expresiones se deben a los ciclos empleados cuando se sale del bucle, en cuyo caso se ejecutan únicamente las dos primeras instrucciones y se introducen los ciclos de espera correspondientes a la dependencia de control generada por la segunda, mientras que las 2 instrucciones adicionales que aparecen en el denominador se deben a la ejecución de dichas instrucciones.

e) El tiempo máximo de un acceso de lectura, en caso de no producirse fallo de página, se produciría en el caso de producirse fallo de TLB y fallo de caché:

$$max(T(TLB), T(cache)) + T(TablasPaginas) + T(cache) + T(Mp) = 2 + 3 \times 60 + 2 + 60 = 244ns$$

o lo que es lo mismo 122 ciclos, por lo que se introducirían en el pipeline 121 ciclos de espera.

3 [1,5 puntos] Responda breve y razonadamente a las siguientes cuestiones relativas a los multiprocesadores (MP) y a la computación de altas prestaciones (HPC):

a) ¿Cuál es la ganancia o *speedup* ideal en la versión paralela de un programa ejecutado sobre p procesadores o núcleos?

b) Explique con palabras y sin necesidad de escribir ninguna ecuación lo que establece la Ley de Amdahl. En un proceso cuya versión secuencial durase 8 s y cuya parte paralelizable fuese sólo la mitad de ese tiempo, ¿cuál sería la duración mínima de su versión paralela sobre 4 procesadores?

c) Explique las principales ventajas e inconvenientes de las políticas de invalidación y de actualización para mantener la coherencia de cachés en un MP de memoria compartida.

SOLUCIÓN

a) En una paralelización ideal con todo el trabajo potencialmente mejorable y sin sobrecoste (*overhead*) el tiempo de ejecución original, t, se repartiría uniformente entre los p procesadores o núcleos, obteniéndose un tiempo de t/p y un *speedup* de p, speedup(p) = p

b) En esta ley se dice que la ganancia que se obtiene al paralelizar un programa con p procesadores está limitada por la fracción de ese programa que tiene naturaleza secuencial y que, consecuentemente, no puede paralelizarse. En los extremos, si la fracción mejorable fuese todo el tiempo de ejecución, la ganancia máxima sería p (véase el apartado anterior) y si fuese cero –ninguna fracción podría mejorarse– no se obtendría ganancia (i.e., speedup (p) = 1). En el caso que se plantea, la mitad del tiempo (4 s) no sería mejorable y la otra mitad se ejecutaría en paralelo sobre 4 procesadores o núcleos, por lo que en el caso mejor se repartiría en uniformente entre ellos, significando un tiempo de 1 s en cada procesador y un nuevo tiempo total de ejecución de (4+1) s.

c) La política de invalidación tiene la ventaja de que provoca poco tráfico añadido en el sistema para mantener la coherencia (se difunde la invalidación y mientras no se demande acceso a una de las copias invalidadas no hay más tráfico extra en el sistema). Sin embargo, dependiendo de patrón temporal de acceso al bloque entre los distintos procesadores, si existe compartición de "grano fino" este tráfico debido a los fallos por accesos a bloque invalidados puede ser considerable. Además, su implementación suele resultar más sencilla que la de los algoritmos basados en la política de actualización. En esta última no hay fallos adicionales a los que se producirían en una ejecución sobre un solo procesador –las copias se mantienen siempre coherentes–, pero a cambio aumenta significativamente el tráfico añadido para mantener la coherencia, por lo que resulta más adecuado cuanto más de grano fino sea el patrón temporal de acceso al bloque. También, al contrario de la política anterior, los algoritmos basados en esta política suelen ser de implementación más complicada.

4 [1,5 puntos] Explique el funcionamiento de la instrucción atómica `test&set .Ri, /dir`. Para máquinas de memoria compartida, emplee esta instrucción para implementar la adquisición con espera activa (*spin-lock*) de un cerrojo. Indique cómo se comportaría la implementación propuesta en un sistema de memoria compartida con cachés privadas y en el que se utilizase *actualización* como política de coherencia.

SOLUCIÓN

La instrucción devuelve en Ri el valor previo de la dirección especificada, dir, y escribe un uno en esta posición y esto de manera atómica, i.e., con la garantía de que ningún otro procesador pueda acceder a esta misma dirección mientras dura la ejecución completa del t&s. Este comportamiento lo podemos expresar de la siguiente forma:

```
Test:  test&set  .Ri, /dir
       {
        tmp <-- /dir
        /dir <-- '1'
        Ri <-- tmp
       }
```

supuesto que las acciones comprendidas entre las llaves ocurren atómicamente.

Una posible implementación de una primitiva de adquisición de un cerrojo en la dirección lock podría ser:

```
Test:  test&set  .R1, /lock
       bnz test
       ret
```

que es una implementación con espera activa en la que en cada "vuelta" se fuerza a uno el valor del cerrojo. En la estructura que se plantea, cada procesador tendrá copia del bloque que contiene la dirección del cerrojo y, consecuentemente, la política de actualización hará que a cada vuelta al bucle de espera sean actualizadas las copias de los demás procesadores, por lo que se evitarían los ulteriores fallos de caché a lo que daría lugar el uso de la política de invalidación. Se trata de un buen ejemplo de la citada compartición de grano fino.

NOTAS: 23 de julio de 2012 DURACIÓN: 2h 30 minutos
REVISIÓN: 24 de julio de 2012 PUNTUACIÓN: Se indica en cada apartado